

Engenharia de Software II

UNIDADE CURRICULAR	ENGENHARIA DE SOFTWARE II
PERÍODO LETIVO: 3º SEMESTRE	CARGA HORÁRIA TOTAL: 60hs
PRÉ-REQUISITO	Engenharia de Software I e Programação Orientada a Objetos
CARACTERIZAÇÃO DA UNIDADE CURRICULAR	
Compreensão das principais arquiteturas de software e padrões de projeto e sua aplicação no desenvolvimento de soluções de software modernas e aderentes aos seus requisitos.	
COMPETÊNCIA ESSENCIAL	
Projetar soluções de software em conformidade com seus requisitos, considerando as principais arquiteturas e padrões de projetos utilizados em sistemas modernos e escaláveis.	
ELEMENTOS DE COMPETÊNCIA - COMPETÊNCIAS RELACIONADAS	
Compreender o papel do arquiteto de software no processo de desenvolvimento de software.	
Propor soluções utilizando arquiteturas de software aderentes aos seus requisitos.	
Compreender a cultura DevOps e suas implicações no contexto do desenvolvimento de software.	
Utilizar padrões de projeto no desenvolvimento de soluções de software.	
Documentar adequadamente arquiteturas e projetos de software utilizando modelos padrão de mercado.	
BASES TECNOLÓGICAS	
O papel do Arquiteto de Software.	
Arquiteturas de Software: teoria e prática.	
Código limpo.	
Padrões de Projeto: teoria e prática.	
Diagrama de sequência, de componentes e de implantação.	
Controle de versionamento de software.	
DevOps.	
Integração e entrega contínua.	
BIBLIOGRAFIA BÁSICA	
BECK, Kent. Padrões de Implementação – Um Catálogo de Padrões indispensáveis para o dia a dia do Programador. Bookman. 2013.	
GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. Padrões de Projetos . Grupo A, 2011.	
MARTIN, Robert. Código Limpo – Habilidades Práticas do Agile Software. Edição revisada. Alta Books. 2011.	
MARTIN, Robert. Arquitetura Limpa – O Guia do Artesão para Estrutura e Design de Software. Alta Books. 2019.	
PRESSMAN, Roger S. Engenharia de software : uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016.	
BIBLIOGRAFIA COMPLEMENTAR	
FOWLER, Martin. UML essencial : um breve guia para a linguagem-padrão de modelagem de objetos. 3. ed. Porto Alegre: Bookman, 2005.	
MARTIN, Robert. O Codificador Limpo - Um Código de Conduta para Programadores Profissionais. Alta Books. 2012.	
BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. UML : guia do usuário. 2. ed. rev. atual. Rio de Janeiro: Elsevier, 2012.	
PAULA FILHO, Wilson de Pádua. Engenharia de software : fundamentos, métodos e padrões. 3. ed. Rio de Janeiro: LTC, 2009.	
RUMBAUGH, James et al. Modelagem e projetos baseados em objetos . Rio de Janeiro: Elsevier, 2004.	

A photograph showing a person's legs and feet as they sweep a dark wood floor. A large pile of dark dust or debris is visible on the floor. The person is wearing light-colored trousers and dark shoes.

Clean Code

Referência

Robert C. Martin Series

Clean Code

A Handbook of Agile Software Craftsmanship

Robert C. Martin



Foreword by James O. Coplien



MAS PORQUE?

Reflexão

Abrir mão de qualidade e processos em nome da velocidade?

Reflexão

Abrir mão de qualidade e processos em nome da velocidade?

Se a tua lancheria preferida abrir mão da higiene para entregar seu lanche mais rapidamente, tudo bem para ti?

Não realizar a higienização dos equipamentos cirúrgicos para realizar mais procedimentos por dia, pode?

Reflexão

Abrir mão de qualidade e processos em nome da velocidade?

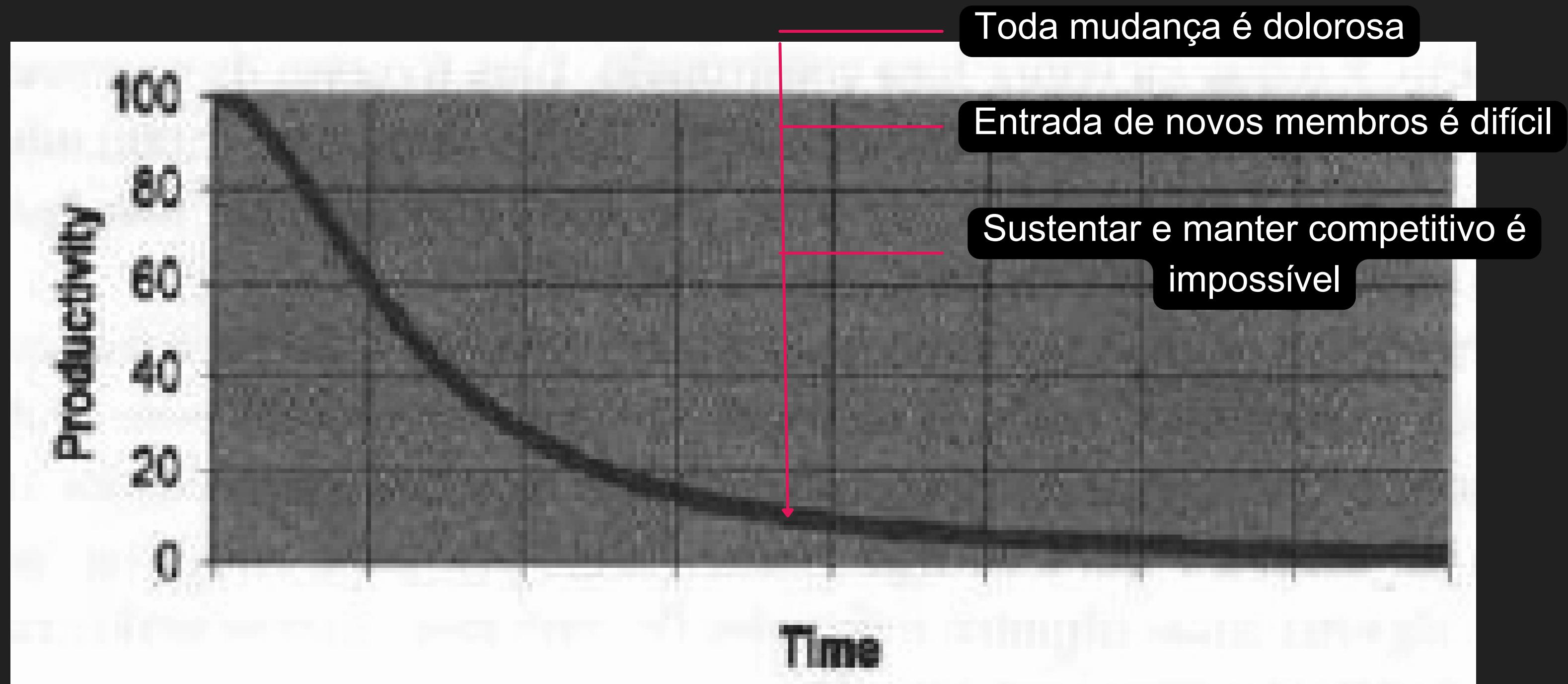
**Se a tua lancheria preferida
abrir mão da higiene para
entregar seu lanche mais
rapidamente, tudo bem para
ti?**

E se o dono da
lancheria pressionar
que precisa entregar
mais lanches por dia?

E se o Diretor do
hospital insistir que
precisa liberar leitos?

**Não realizar a higienização dos
equipamentos cirúrgicos para
realizar mais procedimentos
por dia, pode?**

Relação tempo/produtividade



Estatística

Um dev experiente passa 10% do seu tempo codando, 90% lendo.

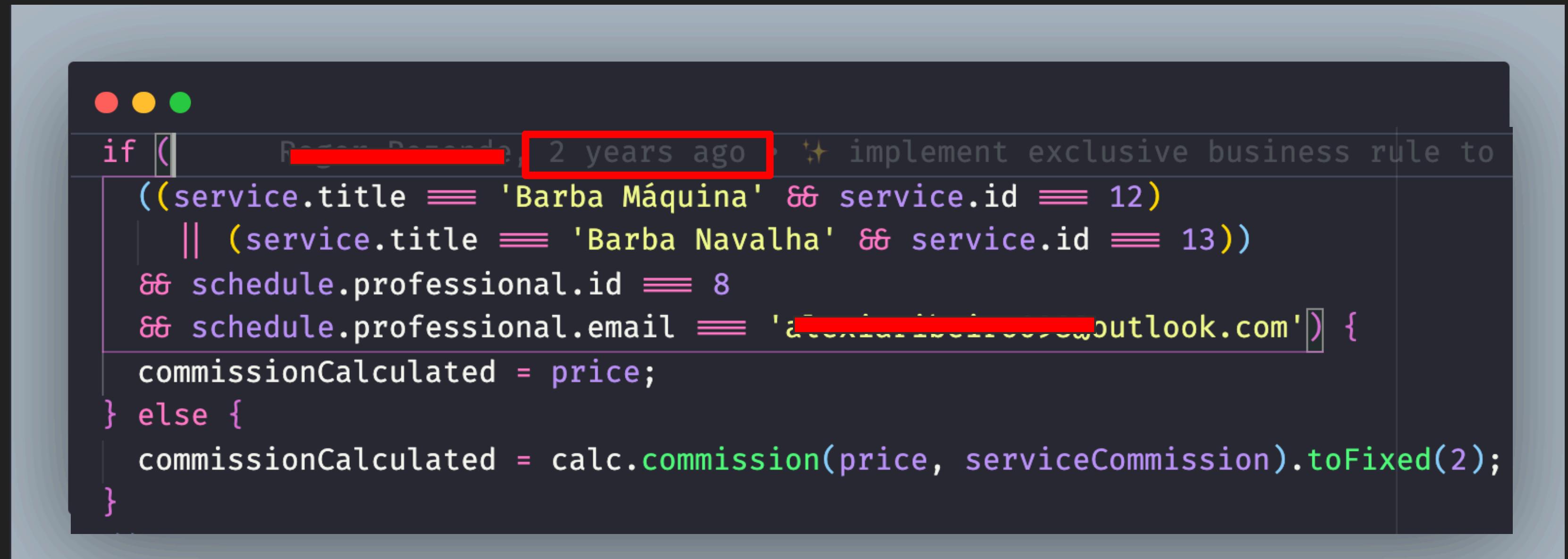
Aquela decisão...

"Faz essa gambiarra para entregar que depois a gente arruma."

```
● ● ●  
1 if (  
2   ((service.title === 'Barba Máquina' && service.id === 12)  
3     || (service.title === 'Barba Navalha' && service.id === 13))  
4   && schedule.professional.id === 8  
5   && schedule.professional.email === 'alexianibaino@outlook.com') {  
6     commissionCalculated = price;  
7   } else {  
8     commissionCalculated = calc.commission(price, serviceCommission).toFixed(2);  
9   }
```

Aquela decisão...

Lei de LeBlanc: "Mais tarde é igual a nunca".



```
● ● ●
if (      Roger Resende, 2 years ago ✨ implement exclusive business rule to
    ((service.title == 'Barba Máquina' && service.id == 12)
     || (service.title == 'Barba Navalha' && service.id == 13))
     && schedule.professional.id == 8
     && schedule.professional.email == 'roger.resende.outlook.com') {
    commissionCalculated = price;
} else {
    commissionCalculated = calc.commission(price, serviceCommission).toFixed(2);
}
```

The screenshot shows a code editor with a dark theme. At the top left, there are three colored dots (red, yellow, green). The main area contains a snippet of Java-like pseudocode. A red rectangular box highlights the timestamp '2 years ago' in the first line. The code itself is a conditional statement that checks if a service title is either 'Barba Máquina' or 'Barba Navalha', if its id is 12 or 13, and if the professional's id is 8 and email is a specific Outlook address. It then calculates a commission based on the price and a commission percentage.

Variáveis

Melhorar a nossa leitura de código começa pelo básico - as nossas variáveis.

Variáveis

Uma tarefa "simples" e importante como dar nome a um filho.



Mais variáveis

Dê significado a elas



```
1
2 // Vish
3 const x = 30;
4
5 // Putz
6 const duration = 30;
7
8 // Bem melhor
9 const durationInMinutes = 30;
10
11 }
```

Nomes significativos



Nomes de variáveis

Nomes de variáveis, devem ser bem pensados, para VOCÊ e o PRÓXIMO dev a tocar no código.

As variáveis precisam ser auto documentadas, sem a necessidade de interpretação.

Nomes de variáveis

// ??

const genymdhms;

Nomes de variáveis

// Generation Year Month Day Hours Minutes and Seconds

const genymdhms;

Nomes de variáveis

// Generation Year Month Day Hours Minutes and Seconds ✗

const genymdhms; ✗

const generationTimestamp; ✓

Exemplo de Nomes de variáveis ruins

```
● ● ●

1 createRestriction: async (parent, { data }, { db }) => {
2   if (!isTime(data.time_start)
3     || !isTime(data.time_end)
4   ) {
5     throw new Error('Horário inválido');
6   }
7
8   const time1 = dayjs().utc().local().format(`1999-01-01 ${data.time_start}`);
9   const time2 = dayjs().utc().local().format(`1999-01-01 ${data.time_end}`);
10
11  if (time1 > time2) {
12    throw new Error('Horário inválido');
13  }
```

Simples ajuste que poderia melhorar a leitura

```
● ● ●  
1  createRestriction: async (parent, { data }, { db }) => {  
2    if (!isTime(data.time_start)  
3      || !isTime(data.time_end)  
4    ) {  
5      throw new Error('Horário inválido');  
6    }  
7  
8    const timeStart = new Date(`1999-01-01 ${data.time_start}`);  
9    const timeEnd = new Date(`1999-01-01 ${data.time_end}`);  
10  
11   if (timeStart > timeEnd) {  
12     throw new Error('Horário inválido');  
13   }
```

KISS

Keep It Stupid Simple

Escrever código é como escrever um livro, ele tem que ser compreensível e não complexo e ilegível

Regra de escoteiro

Deixe o ~~acampamento~~**código** mais limpo do que você encontrou.

Tente sempre aplicar melhorias nos códigos que estão mantendo, ou no mínimo mantenha a qualidade

Um case

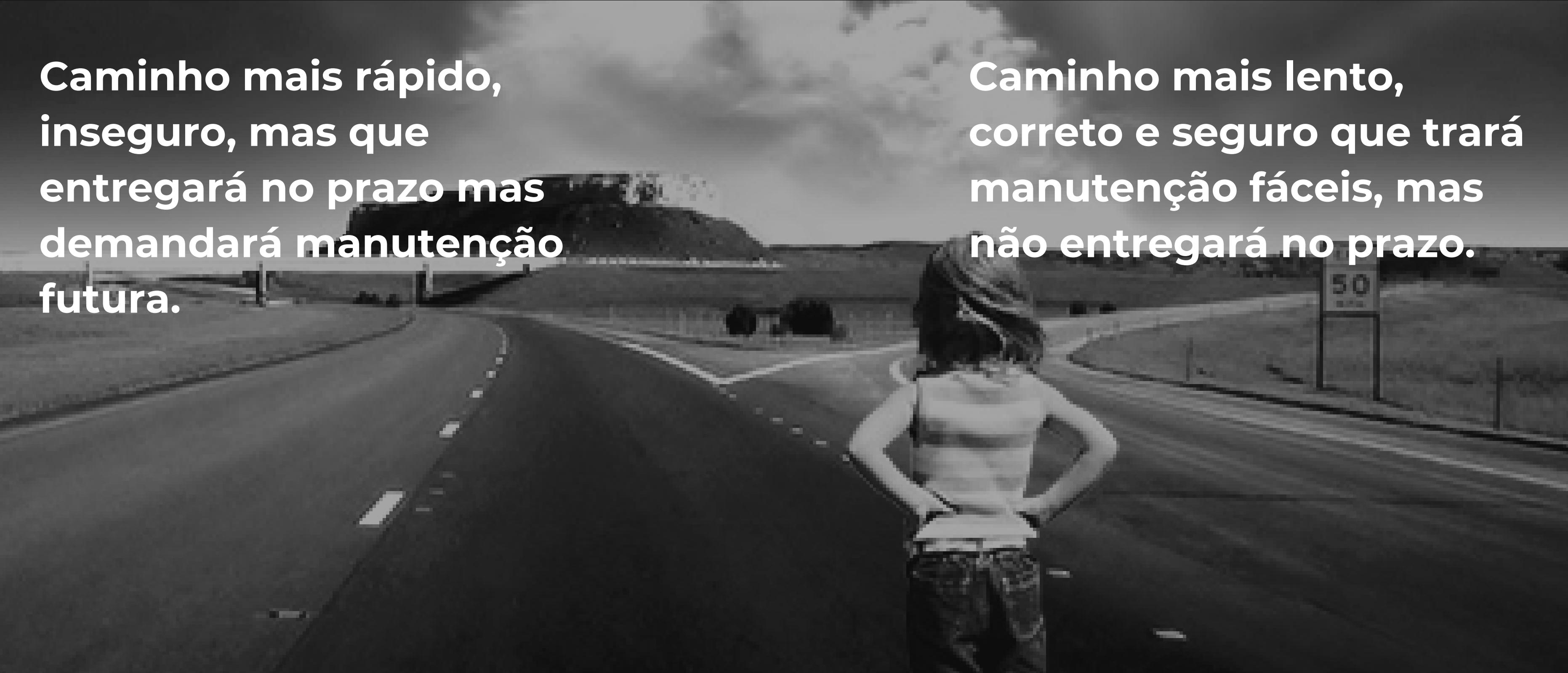
"Temos uma entrega em dois dias e precisamos de uma recomendação de conteúdo!"

JOACIL LUIS DE OLIVEIRA

A VERDADEIRA
ORIGEM
DO MAL



Reflexão



**Caminho mais rápido,
inseguro, mas que
entregará no prazo mas
demandará manutenção
futura.**

**Caminho mais lento,
correto e seguro que trará
manutenção fáceis, mas
não entregará no prazo.**

Conteúdos recomendados



```
● ● ●  
1  async getRecommendedContents(  
2    feelingId: string,  
3    userId: string,  
4  
5    query: ContentDTOQuery,  
6  ): Promise<Content[]> {  
7    const user = await this.userService.findById(userId);  
8    const data = await this.contentRepository  
9      .createQueryBuilder('content')  
10     .select('content.id, content.title, content.type')  
11     .leftJoin('content.categories', 'categories')  
12     .leftJoin('categories.feelings', 'feelings')  
13     .leftJoinAndSelect('content.thumbnail', 'thumbnail')  
14     .where('feelings.id = :feelingId', { feelingId })  
15     .limit(query.paging.limit)  
16     .orderBy('content.created_at', 'DESC')  
17     .offset(query.paging.offset);  
18    if (!user.subscriber) {  
19      data.andWhere('content.premium = false');  
20    }  
21  
22    return data.getRawMany();  
23  }
```

Conteúdos recomendados

Análise

Query na service que recomenda conteúdos com base em como o usuário está se sentindo.

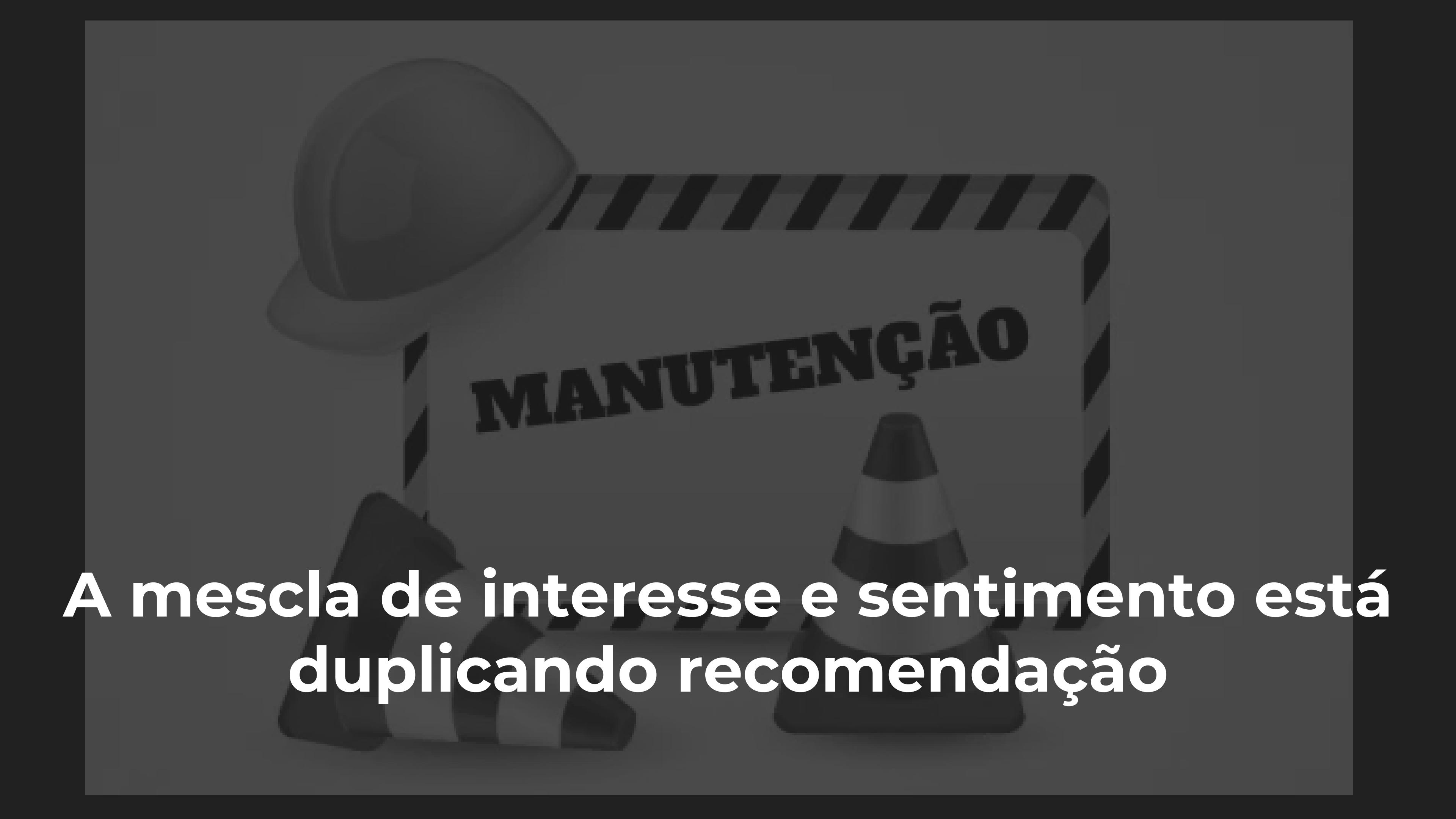
```
● ● ●  
1  async getRecommendedContents(  
2    feelingId: string,  
3    userId: string,  
4  
5    query: ContentDTOQuery,  
6  ): Promise<Content[]> {  
7    const user = await this.userService.findById(userId);  
8    const data = await this.contentRepository  
9      .createQueryBuilder('content')  
10     .select('content.id, content.title, content.type')  
11     .leftJoin('content.categories', 'categories')  
12     .leftJoin('categories.feelings', 'feelings')  
13     .leftJoinAndSelect('content.thumbnail', 'thumbnail')  
14     .where('feelings.id = :feelingId', { feelingId })  
15     .limit(query.paging.limit)  
16     .orderBy('content.created_at', 'DESC')  
17     .offset(query.paging.offset);  
18    if (!user.subscriber) {  
19      data.andWhere('content.premium = false');  
20    }  
21  
22    return data.getRawMany();  
23  }
```

Queremos agora que a recomendação
leve em consideração também os
interesses do usuário

Solução rápida

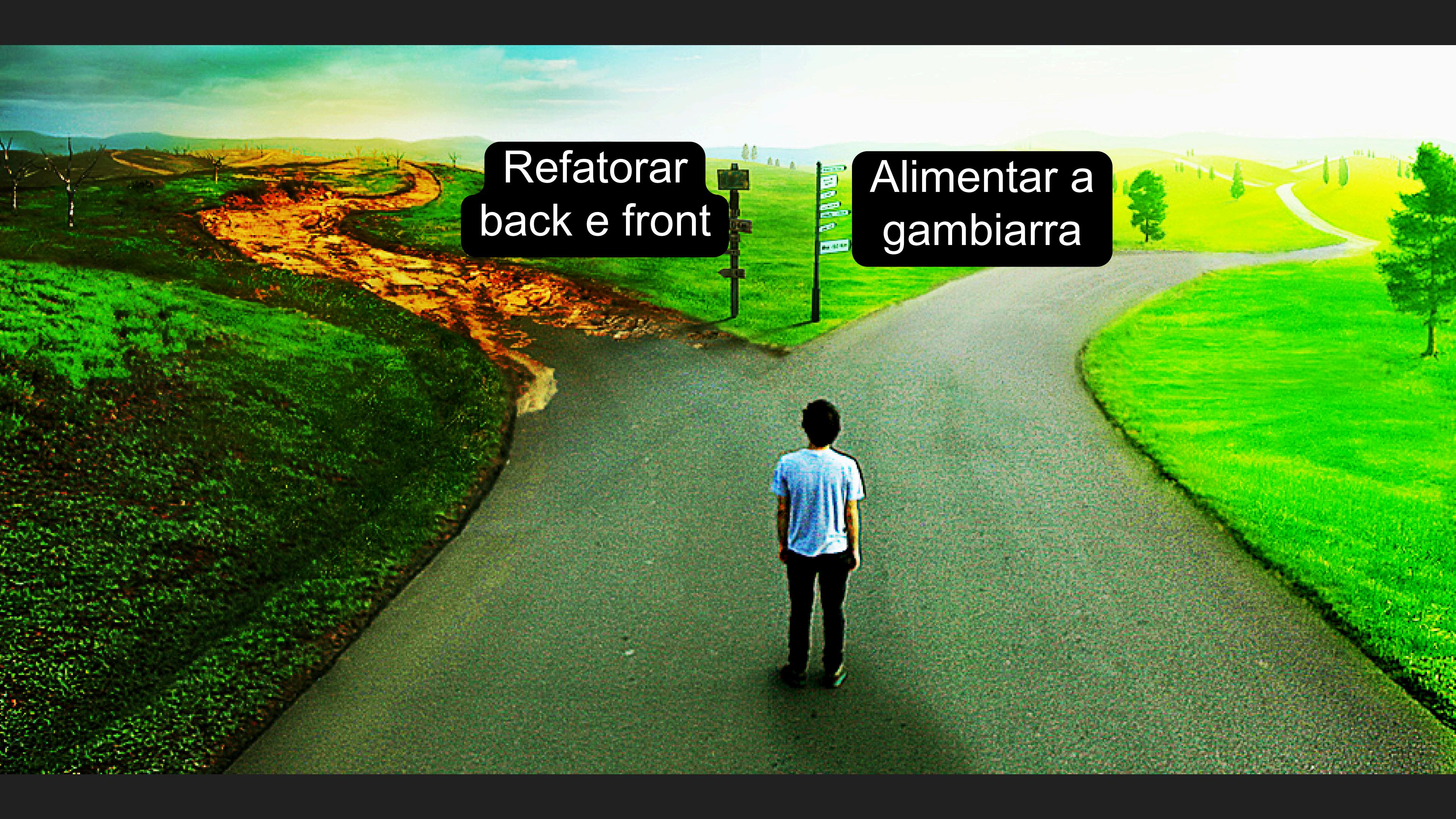
HowIFeelMain/in dex.tsx

```
● ● ●  
1  useEffect(() => {  
2      if (feeling && feeling.id) {  
3          fetchRecommendedByFeeling({  
4              variables: {  
5                  feelingId: feeling.id,  
6                  paging: {  
7                      limit: maxRecommendedArraySize,  
8                  },  
9              },  
10         });  
11     }  
12 }, [feeling, fetchRecommendedByFeeling]);  
13  
14 useEffect(() => {  
15     if (user?.categories) {  
16         fetchRecommendedByCategories({  
17             variables: {  
18                 categories: user.categories.map((category) => category.id),  
19                 paging: {  
20                     limit: maxRecommendedArraySize,  
21                 },  
22             },  
23         });  
24     }  
25 }, [user, fetchRecommendedByCategories]);
```



MANUTENÇÃO

A mescla de interesse e sentimento está
duplicando recomendação



Refatorar
back e front

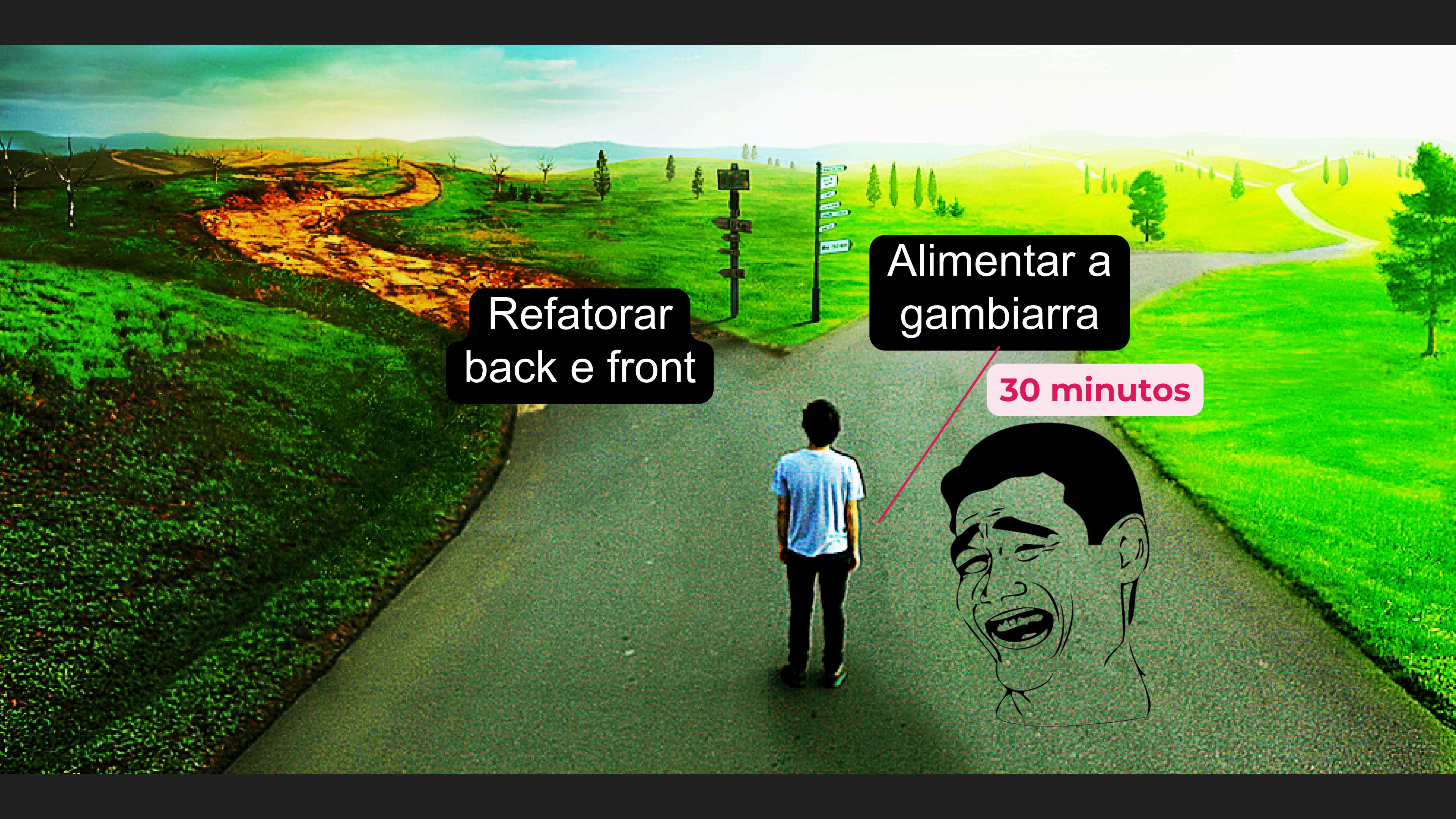
Alimentar a
gambiarra



Refatorar
back e front

Alimentar a
gambiarra

20 horas



Refatorar
back e front

Alimentar a
gambiarra

30 minutos



How I Feel Main

index.tsx

```
● ● ●
1 // O código abaixo faz um merge dos 2 tipos de conteudos recomendados em um array de tamanho máximo
2 // a lógica é popular um array com o maior numero possivel de conteúdo, balanceadamente
3 // balanceadamente significa que se tiver menos conteudos de um tipo ele vai popular com mais conteudos de outro tipo
4
5 const contentsByFeeling = contentsByFeelingData?.recommendedContents ?? [];
6
7 // O Set abaixo armazena todos os IDs de contentsByFeelingData,
8 // logo em seguida esse Set é utilizado para filtrar todos o items com o
9 // mesmo ID em contentsByCategoriesData
10 // Items com o mesmo ID não serão armazenado em contentsByCategories
11 // e por consequencia não serão usados na logica de balanceamento
12 const contentsByFeelingSet = new Set(
13   contentsByFeeling.map((item) => item.id),
14 );
15 const contentsByCategories =
16   contentsByCategoriesData?.contents.filter((item) => {
17     return !contentsByFeelingSet.has(item.id);
18   }) ?? [];
19
20 let contentsByFeelingMaxSize =
21   maxRecommendedArraySize / numberOfDifferentRecommendedContents;
22 let contentsByCategoriesMaxSize =
23   maxRecommendedArraySize / numberOfDifferentRecommendedContents;
24
25 // logica pra decidir o tamanho maximo do array de conteúdos recomendados pela categoria
26 // se os conteúdos recomendados pelos feelings for menor do que a metade do tamanho maximo do array, significa que
27 // o tamanho maximo do array de conteudos recomendados pela categoria pode ser maior
28 if (contentsByCategoriesData) {
29   const contentsByCategoriesSize = contentsByCategories.length;
30   if (contentsByCategoriesSize < contentsByCategoriesMaxSize - 1) {
31     contentsByFeelingMaxSize =
32       maxRecommendedArraySize - contentsByCategoriesSize;
33   }
34 }
35
36 // mesma logica que acima, mas o contrário (utilizando o arrau de conteudos recomendados pelos feelings)
37 if (contentsByFeelingData) {
38   const contentsByFeelingSize = contentsByFeeling.length;
39   if (contentsByFeelingSize < contentsByFeelingMaxSize - 1) {
40     contentsByCategoriesMaxSize =
41       maxRecommendedArraySize - contentsByFeelingSize;
42   }
43 }
44
45 // com a regra abaixo conteúdos baseados nos feelings vem antes que os baseados em categorias. (achei que provavelmente seria assim)
46 const resultRecommendedArray = [
47   ...contentsByFeeling.slice(0, contentsByFeelingMaxSize),
48   ...contentsByCategories.slice(0, contentsByCategoriesMaxSize),
49 ];
```

Evite os bugs bumerangues

Crie sempre que possível, testes que validam suas correções.



**Criando testes para suas correções,
você garante que um próximo
desenvolvedor não descarte seu
código por considerar "sem sentido"**

Condicionais e repetições

Exemplo de Condicional



```
1 if (foundUser.validationCode === code || code === 9999) {  
2   const buildUser: User = this.userRepository.create(data);  
3   if (buildUser.responsibleContact) {  
4     buildUser.responsibleContact = JSON.parse(  
5       JSON.stringify(buildUser.responsibleContact),  
6     );  
7   }  
8   buildUser.validationCode = null;  
9   const saved = await this.userRepository.save({  
10     ...foundUser,  
11     ...buildUser,  
12   });  
13  
14   return saved;  
15 }
```

Exemplo de Condicional

Melhorando leitura

Condicionais e repetições, sempre que precisar existir, que tenha uma linha só



```
1 if (foundUser.validationCode === code || code === 9999) {  
2     this.fixAndSaveUser(foundUser);  
3 }
```

Exemplo de Condicional Refatorando condição

Extrair condições para variáveis facilitam
muito a leitura de código

```
● ● ●  
1 const isValidCode = foundUser.validationCode === code || code === 9999;  
2 if (isValidCode) {  
3   this.fixAndSaveUser(foundUser)  
4 }
```

Funções

Más e melhores práticas

Evite funções grandes

Criar código complexo não é bonito, muito menos certo. Organize suas funções/métodos chamando outras funções internas, assim como os capítulos dos livros possuem seções.

Seu código ficará mais agradável e o leitor mais confiante para ler.

Evitar nos métodos e funções

Mais de uma função

Seções

Condicionais

Muitos parâmetros

+ do que 20 linhas

Mais de um nível de abstração

- Alta, média e baixa



Evitar nos métodos e funções

Mais de uma função

Seções

Condicionais

Muitos parâmetros

+ do que 20 linhas

Mais de um nível de abstração

- Alta, média e baixa

```
1  async filterDriverStatisticsByYear(  
2      year: number,  
3  ): Promise<RuleStatisticsTravelDTO[]> {  
4      const DriverStatisticsTravel = [];  
5      let travels;  
6  
7      if (year > 0) {  
8          travels = await this.travelRepository  
9              .createQueryBuilder('travel')  
10             .innerJoin('travel.driver', 'driver')  
11             .innerJoin('driver.identification', 'identification')  
12             .groupBy('identification.name')  
13             .addGroupBy('extract(month from travel.dateTime)')  
14             .select('identification.name')  
15             .addSelect('extract(month from travel.dateTime)', 'month')  
16             .addSelect('count(travel.id)', 'count')  
17             .where('extract(year from travel.dateTime)=:year', { year: year })  
18             .orderBy('month')  
19             .getRawMany();  
20  
21      travels.map((travel) => {  
22          DriverStatisticsTravel.push({  
23              ruleGroup: travel.identification_name,  
24              month: travel.month,  
25              count: travel.count,  
26          });  
27      });  
28  }  
29  return DriverStatisticsTravel;  
30 }
```

Evitar nos métodos e funções

Mais de uma função

Seções

Condicionais

Muitos parâmetros

+ do que 20 linhas

Mais de um nível de abstração

- Alta, média e baixa

```
1  async filterDriverStatisticsByYear(  
2      year: number,  
3  ): Promise<RuleStatisticsTravelDTO[]> {  
4      const DriverStatisticsTravel = [];  
5      let travels;  
6  
7      if (year > 0) {  
8          travels = await this.travelRepository  
9              .createQueryBuilder('travel')  
10             .innerJoin('travel.driver', 'driver')  
11             .innerJoin('driver.identification', 'identification')  
12             .groupBy('identification.name')  
13             .addGroupBy('extract(month from travel.dateTime)')  
14             .select('identification.name')  
15             .addSelect('extract(month from travel.dateTime)', 'month')  
16             .addSelect('count(travel.id)', 'count')  
17             .where('extract(year from travel.dateTime)=:year', { year: year })  
18             .orderBy('month')  
19             .getRawMany();  
20  
21      travels.map((travel) => {  
22          DriverStatisticsTravel.push({  
23              ruleGroup: travel.identification_name,  
24              month: travel.month,  
25              count: travel.count,  
26          });  
27      });  
28  }  
29  return DriverStatisticsTravel;  
30 }
```

Evitar nos métodos e funções

Mais de uma função

Seções

Condicionais

Muitos parâmetros

+ do que 20 linhas

Mais de um nível de abstração

- Alta, média e baixa

```
1  async filterDriverStatisticsByYear(  
2      year: number,  
3  ): Promise<RuleStatisticsTravelDTO[]> {  
4      const DriverStatisticsTravel = [];  
5      let travels;  
6  
7      if (year > 0) {  
8          travels = await this.travelRepository  
9              .createQueryBuilder('travel')  
10             .innerJoin('travel.driver', 'driver')  
11             .innerJoin('driver.identification', 'identification')  
12             .groupBy('identification.name')  
13             .addGroupBy('extract(month from travel.dateTime)')  
14             .select('identification.name')  
15             .addSelect('extract(month from travel.dateTime)', 'month')  
16             .addSelect('count(travel.id)', 'count')  
17             .where('extract(year from travel.dateTime)=:year', { year: year })  
18             .orderBy('month')  
19             .getRawMany();  
20  
21      travels.map((travel) => {  
22          DriverStatisticsTravel.push({  
23              ruleGroup: travel.identification_name,  
24              month: travel.month,  
25              count: travel.count,  
26          });  
27      });  
28  }  
29  return DriverStatisticsTravel;  
30 }
```

Evitar nos métodos e funções

Mais de uma função

Seções

Condicionais

Muitos parâmetros
+ do que 20 linhas

Mais de um nível de abstração

- Alta, média e baixa

O ideal é não ter
parâmetros, mas de 1 a
2 são aceitos e
excepcionalmente 3



```
1  async saveUser(  
2    name: string,  
3    phone: string,  
4    uf: string,  
5    street: string,  
6    number: string,  
7    email: string,  
8    password: string){  
9      // ...  
10 }
```



```
1  async saveUser(  
2    data: CreateUserInput){  
3      // ...  
4 }
```

Evitar nos métodos e funções

Mais de uma função

Seções

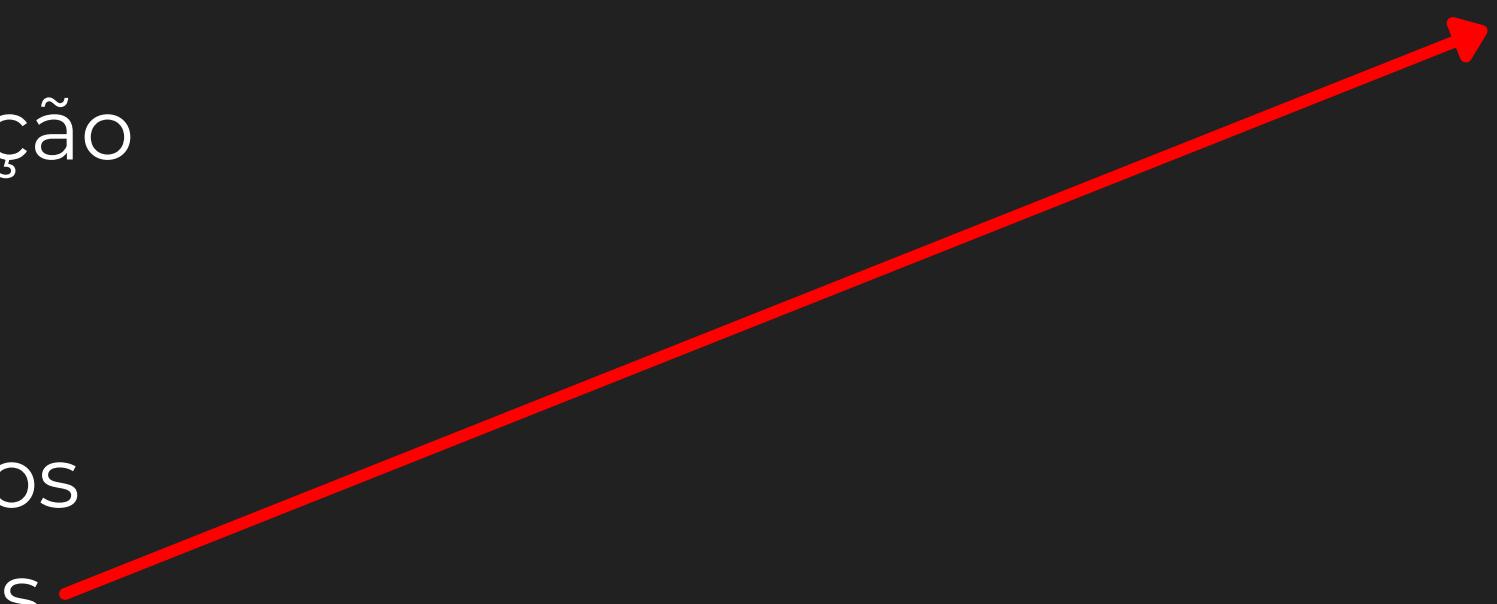
Condicionais

Muitos parâmetros

+ do que 20 linhas

Mais de um nível de abstração

- Alta, média e baixa



```
1  async filterTicketsStatisticsByQuantity(
2    quantity: number,
3  ): Promise<CustomAverageStatisticsTravelDTO> {
4    const TicketsStatisticsTravel = [];
5    let tickets, ticketsMirror;
6    let ticketCount = 0,
7    vehicleCapacity = 0;
8    let ticketCountMirror = 0,
9    vehicleCapacityMirror = 0;
10   tickets = await this.travelRepository
11     .createQueryBuilder('travel')
12     .innerJoin('travel.vehicle', 'vehicle')
13     .innerJoin('travel.tickets', 'tickets')
14     .groupBy('travel.dateTime')
15     .select('travel.dateTime')
16     .addGroupBy('travel.rule')
17     .addSelect('travel.rule')
18     .addGroupBy('travel.cityOrigin')
19     .addSelect('travel.cityOrigin')
20     .addGroupBy('travel.cityDestiny')
21     .addSelect('travel.cityDestiny')
22     .addGroupBy('vehicle.capacity')
23     .addSelect('vehicle.capacity')
24     .addSelect('count(tickets.chairNumber)', 'countTicket')
25     .where('travel.status = :status', { status: 'completed' })
26     .orderBy('travel.dateTime', 'DESC')
27     .limit(quantity)
28     .getRawMany();
29
30   ticketsMirror = await this.travelRepository
31     .createQueryBuilder('travel')
32     .innerJoin('travel.vehicle', 'vehicle')
33     .innerJoin('travel.tickets', 'tickets')
34     .groupBy('travel.dateTime')
35     .select('travel.dateTime')
36     .addGroupBy('travel.rule')
37     .addSelect('travel.rule')
38     .addGroupBy('travel.cityOrigin')
39     .addSelect('travel.cityOrigin')
40     .addGroupBy('travel.cityDestiny')
41     .addSelect('travel.cityDestiny')
42     .addGroupBy('vehicle.capacity')
43     .addSelect('vehicle.capacity')
44     .addSelect('count(tickets.chairNumber)', 'countTicket')
45     .where('travel.status = :status', { status: 'completed' })
46     .orderBy('travel.dateTime', 'DESC')
47     .offset(quantity)
48     .limit(quantity)
49     .getRawMany();
50
51   tickets.map((ticket) => {
52     ticketCount += Number(ticket.countTicket);
53     vehicleCapacity += Number(ticket.vehicle_capacity);
54     TicketsStatisticsTravel.push({
55       dateTime: ticket.travel_dateTime,
56       cityOrigin: ticket.travel_cityOrigin,
57       cityDestiny: ticket.travel_cityDestiny,
58       state: 'busy',
59       quantity: ticket.countTicket,
60       ruleType: ticket.travel_rule,
61     });
62   );
63   TicketsStatisticsTravel.push({
64     dateTime: ticket.travel_dateTime,
65     cityOrigin: ticket.travel_cityOrigin,
66     cityDestiny: ticket.travel_cityDestiny,
67     state: 'total',
68     quantity: ticket.vehicle_capacity,
69     ruleType: ticket.travel_rule,
70   });
71 }
72
73   ticketsMirror.map((ticket) => {
74     ticketCountMirror += Number(ticket.countTicket);
75     vehicleCapacityMirror += Number(ticket.vehicle_capacity);
76   });
77
78   if (vehicleCapacity <= 0 || vehicleCapacityMirror <= 0) {
79     throw new NotFoundException(TRAVEL_INSUFFICIENT_DATA);
80   }
81
82   const average = (
83     (ticketCount / vehicleCapacity -
84     ticketCountMirror / vehicleCapacityMirror) *
85     100
86   ).toFixed(1);
87   return { ...{ TicketsStatisticsTravel: TicketsStatisticsTravel }, average };
88 }
```

Evitar nos métodos e funções

Mais de uma função
Seções

Condicionais

Muitos parâmetros
+ do que 20 linhas

Mais de um nível de abstração

- Alta, média e baixa



```
1  async validateUser(id: string, code: number): Promise<Status> {
2    const name: string;
3    const code: string;
4    const email: string;
5
6    const foundUser: User = await this.userRepository.findOne(id);
7    if (!foundUser) {
8      throw new NotFoundException(consts.USER_NOT_FOUND);
9    }
10   if (foundUser.validationCode === code || code === 2504) {
11     foundUser.status = Status.ATIVO;
12   } else {
13     throw new UnauthorizedException(consts.USER_INVALID_CODE);
14   }
15   const updatedUser = await this.userRepository.save(foundUser);
16
17   return updatedUser.status;
18 }
```

Evitar nos métodos e funções

- Executar comandos e consultas ao mesmo tempo.

Funções/Métodos deve fazer ou responder algo, **mas não** as duas coisas ao mesmo tempo

Evitar nos métodos e funções

Executar comandos e consultas ao mesmo tempo.

Qual código é o bem escrito?

```
● ● ●  
1 class CarrinhoDeCompras {  
2     private itens: string[] = [];  
3  
4     adicionarItemEObterTotal(itens: string[]): number {  
5         this.itens.push(...itens);  
6         return this.itens.length;  
7     }  
8 }  
9  
10 const carrinho = new CarrinhoDeCompras();  
11 console.log(carrinho.adicionarItemEObterTotal(["banana", "maçã"]));
```

```
● ● ●  
1 class CarrinhoDeCompras {  
2     private itens: string[] = [];  
3  
4     adicionarItem(itens: string[]): void {  
5         this.itens.push(...itens);  
6     }  
7  
8     obterTotalDeItens(): number {  
9         return this.itens.length;  
10    }  
11 }  
12  
13 const carrinho = new CarrinhoDeCompras();  
14 carrinho.adicionarItem(["banana", "maçã"]);  
15 console.log(carrinho.obterTotalDeItens());
```

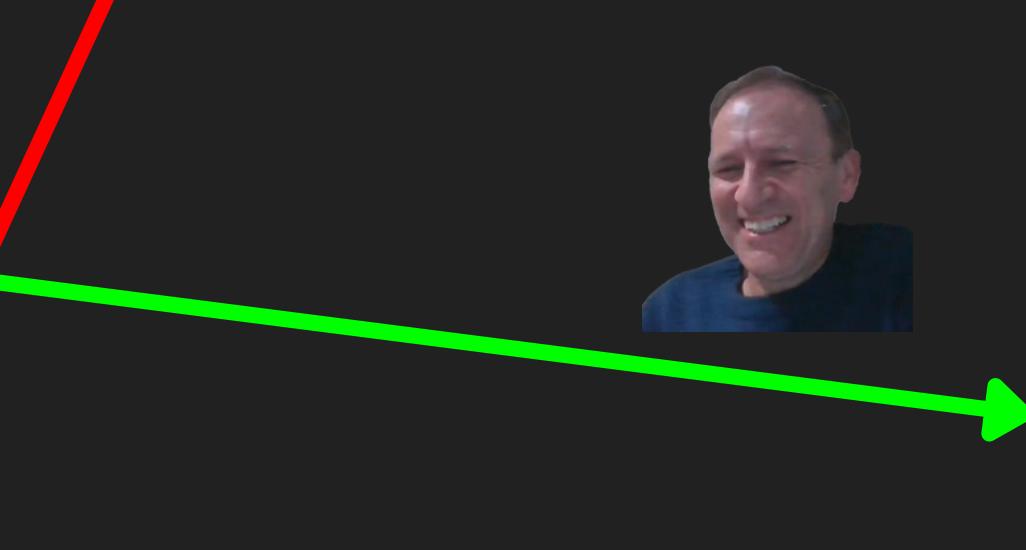
Evitar nos métodos e funções

- Retornar códigos de erros



```
● ● ●  
1 if(deletePage(page) == Status.OK) {  
2   if(registy.deleteReference(page.name) == Status.OK) {  
3     if(configKeys.deleteKey(page.name.makeKey()) == Status.OK) {  
4       console.log('Página excluída');  
5     } else{  
6       console.log('configkey não pode ser excluída');  
7     }  
8   } else{  
9     console.log('DeleteReference não pode ser excluído');  
10 }  
11 } else{  
12   console.log('Exclusão de page falhou');  
13 }  
14 }
```

Utilizar exceções no lugar de códigos de erro evitam aninhamento de código e a **sobrecarga do chamador**



```
● ● ●  
1 try {  
2   deletePage(page);  
3   registy.deleteReference(page.name);  
4   configKeys.deleteKey(page.name.makeKey())  
5 } catch (error) {  
6   console.log(error.message);  
7 }
```